CHALMERS
UNIVERSITY OF TECHNOLOGY

Reinforcement Learning with Temporal Logic Constraints

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 53-4 (2020) 485–492

Reinforcement Learning with Temporal Logic Constraints ⋆

Bengt Lennartson * **Qing-Shan Jia** **

* *Division of Systems and Control, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (e-mail: bengt.lennartson@chalmers.se).*
** *CFINS, Department of Automation, BNRist, Tsinghua University, Beijing 100084, China (e-mail: jiaqs@tsinghua.edu.cn).*

Abstract: Reinforcement learning (RL) is an agent based AI learning method, where learning and optimization are combined. Dynamic programming is then performed iteratively, based on reward and next state observations from the system to be controlled. A brief survey of RL is given, followed by an evaluation of a recently proposed method to include temporal logic safety and liveness guarantees in RL, here combined with classical performance optimization. RL is based on Markov decision processes (MDPs), and to reduce the number of observations from the system, a modular MDP framework is proposed. In the learning process, it is then assumed that some parts of the system are represented by known MDP models, while other parts can be estimated by observations from the real system. Local information from the modular system may then be used to reduce the computational complexity, especially in the handling of safety properties.

Keywords: reinforcement learning, adaption, temporal logic specifications, modular systems

1. INTRODUCTION

Formal verification and optimization of probabilistic systems are often performed based on MDPs (Baier and Katoen, 2008; Puterman, 2014). An MDP is a dynamic model including a number of parameters, where especially the parameters of the transition probability and reward functions are often unknown or at least uncertain. When a controller is designed based on an MDP, it is then natural to include a learning procedure, where a control policy is determined online based on data from the system to be controlled.

In *adaptive control* such a learning procedure is included, where often a model of the plant to be controlled is estimated by for instance recursive least square estimation (Astrom and Wittenmark, 2008). Based on the estimated plant model, a control policy is then designed on-line. When the plant dynamics changes, the estimated plant model is also updated and the control policy is adapted to the modified plant dynamics. An alternative to this indirect control policy adaption, where the policy depends on the estimated plant model, is to directly determine a control policy based on online data from the plant. In machine learning such a *direct adaption procedure* has been proposed, called *reinforcement learning* (RL).

RL is a popular agent based AI learning method, where learning and optimization is performed based on a so called Q-function. The Q-function is a modified value function in the optimization that depends on both the system state and possible actions. It is estimated from real or simulated system data, where the next state and resulting reward are observed as a result of a given action, decided by the agent. The optimization of a control policy is performed by a modified dynamic programming procedure for MDPs, where the policy is directly determined by the Q-function (Bertsekas, 2019; Sutton and Barto, 2018; Busoniu et al., 2010; Gosavi, 2015).

RL is a black box learning method, where there are no guarantees on logical properties of the resulting controlled system. Recently, some methods have been proposed where the closed loop behavior also includes temporal logic guarantees. Sadigh et al. (2014) proposed a transformation of linear temporal logic (LTL) properties to an MDP optimization problem, based on a Rabin automaton and solved by Q-learning. Recently, the Rabin automaton with its double exponential complexity and complex acceptance condition was replaced by the Limit Deterministic Büchi Automaton (LDBA) in (Hasanbeig et al., 2019a), mainly motivated by the simplified acceptance condition. In (Hasanbeig et al., 2019b) this procedure was extended to also include uncertainties in the state labelling function.

This model-free Q-learning procedure is in this paper restricted to LTL formulas with corresponding deterministic Büchi automata, which further simplifies the acceptance condition. It is also argued that most LTL formulas of practical interest can be translated to this deterministic form (Alur and Torre, 2001). This formulation is evaluated in this paper and also combined with ordinary performance optimization. In (Aksaray et al., 2016), Q-learning incorporates signal temporal logic specifications, and Alshiekh et al. (2018) suggest a shield that corrects actions from the agent, if they violate safety specifications.

A basic problem with the model-free Q-learning procedure is the requirement of large data sets. The solution to this problem

⋆ This work was supported by The Swedish Foundation for Strategic Research, through the Smart Assembly 4.0 project, within the Winquist Laboratory; SyTec – Systematic Testing of Cyber-Physical Systems, a Swedish Science Foundation grant for strong research environment; Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation; NSFC 61673229 and the 111 International Collaboration Project of China (No. BP2018006). The support is gratefully acknowledged.

10.1016/j.ifacol.2021.04.044

is generally to utilize as much system information as possible. A modular approach is therefore suggested in this paper, where local learning and analysis can reduce the computational complexity. Partial information can for instance be used to generate safety restrictions by local synthesis. Models for some parts of the system may be known, while other parts can be estimated, and local abstractions can be performed that still preserve global properties. These examples of local learning and analysis are all based on a modular formulation of MDPs.

This paper contains the following contributions. 1) A brief survey of RL is given, including a short but complete derivation of the model-free Q-learning algorithm. 2) A recently proposed temporal logic and model-free Q-learning procedure is evaluated, including some minor extensions and simplifications. 3) A synchronous composition of modular MDPs is proposed in a flexible setting, handling different types of atomic propositions. 4) It is shown how local analysis, based on partial information of a modular system, can be performed to improve especially the handling of safety guarantees in the optimization procedure.

After this introduction, MDPs and their synchronous composition are defined in Section 2, followed by a brief survey of reinforcement learning in Section 3. Linear temporal logic and related Büchi automata are defined in Section 4, including the Büchi weighted product MDP. A model-free Q-learning procedure including LTL specifications is evaluated in Section 5. Finally, in Section 6 it is shown how this procedure can be extended for modular systems, where model-free and model-based learning are combined. This is especially focussed on local handling of safety properties.

2. MODULAR MARKOV DECISION PROCESSES

A Markov decision process (MDP) can be considered as a transition system with initial and transition probability distributions (Puterman, 2014). Generally, a transition system includes both state and transition labels (Baier and Katoen, 2008). An automaton including marked and forbidden states can therefore be considered as a special case of a transition system, where the state labels are limited to marked and forbidden states.

In the discrete event system community, transition labels are called events (Cassandras and Lafortune, 2008), while in computer science and operation research transition labels are most often called *actions* (Milner, 1989; Baier and Katoen, 2008; Poole and Mackworth, 2017; Puterman, 2014). Also note that in probability theory, an event is a subset of all possible outcomes of a random experiment. Hence, all transition labels in an MDP are called *actions*, although they may not only represent (control) actions but also faults and discrete updates of sensor signals, where the notion event can be seen as a more natural expression.

2.1 MDP with state labels

An MDP is now defined to be able to specify temporal logic properties on state labels.

Definition 1. (Markov Decision Process). A Markov decision process is a tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda, \rho \rangle,$$

where

(i) $\mathcal{S}$ is a countable set of states,

(ii) $\mathcal{A}$ is a finite set of actions, where $\mathcal{A}(s)$ is the set of available actions in state s,

(iii) $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is a transition probability function such that $P(s,a,s')$ is the transition probability for transition (s,a,s') from state s to state s' for action $a \in \mathcal{A}(s)$, where $\sum_{s' \in \mathcal{S}} P(s,a,s') = 1$,

(iv) s_0 is an initial state,

(v) AP is a set of atomic propositions,

(vi) $\lambda : X \to 2^{AP}$ is a state labeling function,

(vii) $\rho : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a reward function, where $\rho(s,a,s')$ is the immediate reward when the transition (s,a,s') is performed. □

The image of the state labeling function $\lambda(s)$ includes those atomic propositions that are satisfied (true) in state s. Thus, the propositional formula

$$\psi_\lambda(s) = \bigwedge_{p \in \lambda(s)} p \ \wedge \bigwedge_{p \in AP \setminus \lambda(s)} \neg p$$

is satisfied in state s. Using the satisfaction relation $\vDash$, this is formally expressed as $s \vDash \psi_\lambda$. If no state label is shown at a state s of a transition system, the default state label is assumed to be $\lambda(s) = \varnothing$.

We also observe that for a *deterministic* MDP, there is only one initial state, and for each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}(s)$, the transition probability $P(s,a,s') = 1$ for only one specific next state s'.

Control policy A control policy determines the action to be taken in each individual state. A memory-less control policy, which only depends on the current state, will be formulated in this paper. This state feedback policy is possible, since desired specifications, in this paper especially temporal logic specifications, are included in the model to be used for control policy design, see Sect. 4.

A control policy for an MDP can be either probabilistic or deterministic. A probabilistic control policy is a mapping $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, where $\pi(s,a)$ determines the probability to take action a in state s, and $\sum_{a \in \mathcal{A}(s)} \pi(s,a) = 1$. A deterministic control policy is simplified to a mapping $\pi : \mathcal{S} \to \mathcal{A}$, where $a = \pi(s)$ is the action to be taken in state s. In this paper it is enough to consider deterministic control policies.

2.2 Synchronization of modular MDPs

To be able to join subsystems, the synchronous composition of two MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ is also defined. This is a minor generalization of Bacci et al. (2013), including a flexible definition of atomic propositions of the composed system $\mathcal{M}_1 \| \mathcal{M}_2$.

The set of atomic propositions AP for $\mathcal{M}_1 \| \mathcal{M}_2$ is obtained by taking the union or intersection of the elements in the individual sets AP_1 and AP_2. More precisely, the propositions in AP are divided into a set of AND-state labels $AP^\wedge$ and a set of OR-state labels $AP^\vee$ such that $AP_i^\wedge \subseteq AP^\wedge$, $AP_i^\vee \subseteq AP^\vee$ and $AP_i^\wedge \dot\cup AP_i^\vee = AP_i$ for $i = 1, 2$. In the synchronous composition, the union is taken on the OR-state labels and the intersection on the AND-state labels.

Forbidden states are examples of OR-state labels, since it is enough that either s_1 or s_2 is forbidden to make the composed

state (s_1, s_2) forbidden. On the other hand, marked (goal) states are examples of AND-state labels, since (s_1, s_2) is only marked if both s_1 and s_2 are marked. Thus, there is a need to distinguish between AND- and OR-state labels in synchronous composition.

Another example is opacity, recently reported in (Noori-Hosseini et al., 2019), where non-safe OR-state labels are used in the current state opacity, while non-safe AND-state labels are used in current state anonymity. AND or alternatively OR synchronization is also proposed for opacity in (Mohajerani and Lafortune, 2019). The formulation here is more flexible, since the OR and AND conditions can be introduced individually on each atomic proposition.

Definition 2. (Synchronous Composition). Let $\mathcal{M}_i = \langle \mathcal{S}_i, \mathcal{A}_i, P_i, s_{0i}, AP_i, \lambda_i, \rho_i \rangle$, $i = 1, 2$, be two MDPs where $AP_i^{\wedge} \subseteq AP^{\wedge}$, $AP_i^{\vee} \subseteq AP^{\vee}$ and $AP_i^{\wedge} \dot{\cup} AP_i^{\vee} = AP_i$ for $i = 1, 2$. The synchronous composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is then defined as

$$\mathcal{M}_1 \| \mathcal{M}_2 = \langle \mathcal{S}_1 \times \mathcal{S}_2, \mathcal{A}_1 \cup \mathcal{A}_2, P, (s_{01}, s_{02}), AP, \lambda, \rho \rangle$$

where

(i) the transition probability $P((s_1, s_2), a, (s_1', s_2'))$

$$= \begin{cases} P_1(s_1, a, s_1') P_2(s_2, a, s_2') & a \in \mathcal{A}_1 \cap \mathcal{A}_2, \\ P_1(s_1, a, s_1') & a \in \mathcal{A}_1 \setminus \mathcal{A}_2, \\ P_2(s_2, a, s_2') & a \in \mathcal{A}_2 \setminus \mathcal{A}_1, \end{cases}$$

(ii) the set of atomic propositions $AP = (AP_1^{\wedge} \cap AP_2^{\wedge}) \cup (AP_1^{\vee} \cup AP_2^{\vee})$,

(iii) the state labeling function $\lambda : \mathcal{S}_1 \times \mathcal{S}_2 \to 2^{AP}$,

(iv) the reward $\rho((s_1, s_2), a, (s_1', s_2'))$

$$= \begin{cases} \rho_1(s_1, a, s_1') + \rho_2(s_2, a, s_2') & a \in \mathcal{A}_1 \cap \mathcal{A}_2, \\ \rho_1(s_1, a, s_1') & a \in \mathcal{A}_1 \setminus \mathcal{A}_2, \\ \rho_2(s_2, a, s_2') & a \in \mathcal{A}_2 \setminus \mathcal{A}_1, \end{cases}$$

□

Observe the two special cases, 1) $\mathcal{A}_1 = \mathcal{A}_2$, also called cross product and then denoted $\mathcal{M}_1 \times \mathcal{M}_2$, and 2) $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$, also called interleaving, where no shared actions are involved.

To be able to synchronize subsystems in arbitrary order is important when modular MDPs are joined, as illustrated and discussed in Sect. 6. The following proposition shows that this is possible.

Proposition 1. The synchronous composition is associative, i.e. $(\mathcal{M}_1 \| \mathcal{M}_2) \| \mathcal{M}_3 = \mathcal{M}_1 \| (\mathcal{M}_2 \| \mathcal{M}_3)$.

Proof: This follows by observing the associativity of the multiplication, union, conjunction and addition operators involved in Def. 2. Note that all atomic propositions in AP are assumed to be divided as $AP^{\vee} \dot{\cup} AP^{\vee}$, where only union operations are performed on the OR-sets and only intersection operations on the AND-sets. □

3. REINFORCEMENT LEARNING

Reinforcement learning (RL) is a learning procedure including optimization based on dynamic programming of MDPs (Bertsekas, 2019; Sutton and Barto, 2018; Busoniu et al., 2010). A brief introduction to RL is given in this section, first assuming that the MDP is known, followed by a model-free online learning procedure.

3.1 Model-based Q-function

The most well known RL procedure is called *Q-learning*, which is based on a reformulation of ordinary dynamic programming for MDPs (Bertsekas, 2019). The value function in the optimization is then extended to not only depend on the current state s, but also on available actions $a \in \mathcal{A}(s)$ in this state. The extended value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, also called the *action-value function*, has given the name to this RL procedure (Sutton and Barto, 2018). First, the relation between dynamic programming and the Q-function is clarified, followed by a model-free Q-learning formulation.

Dynamic programming For a given initial state s_0, different transitions (s_k, a_k, s_{k+1}), $k = 0, 1, 2, \ldots$ will take place, depending on which available actions $a_k \in \mathcal{A}(s_k)$ that are selected. We are searching for a control policy $a_k = \pi(s_k)$, $k = 0, 1, 2, \ldots$ that maximizes the following expected infinite sum of discounted random rewards:

$$V(s_0) = \mathbb{E}\Big[\sum_{k=0}^{\infty} \gamma^k \rho(s_k, a_k, s_{k+1}) \Big].$$

The *discount factor* γ, where $0 < \gamma < 1$, determines how far ahead rewards should influence a control policy. For an arbitrary state s and a given policy π, the *Bellman Equation* is

$$V^{\pi}(s) = \sum_{s' \in \mathcal{S}} P(s, a, s') [\rho(s, a, s') + \gamma V^{\pi}(s')].$$

The *Bellman Optimality Equation* shows that the optimal state-value function V^* should satisfy for each $s \in \mathcal{S}$

$$V^*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s, a, s') \big[\rho(s, a, s') + \gamma V^*(s')\big].$$

This expression is the core of dynamic programming for MDPs (Bertsekas, 2019) .

Q-function For a state-action pair (s, a), the Q-function is now defined as

$$Q(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') \big[\rho(s, a, s') + \gamma V^*(s')\big].$$

This means that the Bellman Optimality Equation equation can be reformulated as $V^*(s) = \max_{a \in \mathcal{A}(s)} Q(s, a)$, and the Q-function can be expressed as

$$Q(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') \big[\rho(s, a, s') + \gamma \max_{a' \in \mathcal{A}(s')} Q(s', a')\big]. \tag{1}$$

Thus, for each state s the optimal control policy is

$$\pi(s) = \arg \max_{a \in \mathcal{A}(s)} Q(s, a). \tag{2}$$

The dependency on available actions in $Q(s, a)$ means that the control policy π is immediately achieved by the maximum operator. This is a nontrivial task when only the optimal value function V^* is known. Apart from that, the introduction of the action-value function Q has only resulted in an alternative formulation of the dynamic programming problem.

The following subsections show that a model-free optimization can also be formulated based on the Q-function. This formulation does not include the transition probability and reward functions P and ρ.

3.2 Q-learning for deterministic systems

To obtain a model-free Q-learning formulation, a deterministic model is first considered. It means that the transition probability $P(s,a,s')=1$ for one specific next state s'. Thus, the probability distribution is simplified to one unique transition (s,a,s'), and the Q-function in (1) is simplified to

$$Q(s,a)=\rho(s,a,s')+\gamma \max_{a'\in\mathcal{A}(s')} Q(s',a').$$

Since $Q(s,a)$ depends on itself at the next state ($Q(s',a')$), an iterative procedure is required to determine this Q-function. An estimate is then normally initiated as $\widehat{Q}_0(s,a)=0$ for all state-action pairs (s,a).

When the action a is sent to the controlled system in its current state s, the resulting next state s' and resulting reward r' can be observed and fed back to the control agent. Given this information, i.e. the tuple (s,a,s',r'), the estimate of the Q-function can be updated as

$$\widehat{Q}_{k+1}(s,a)=r'+\gamma \max_{a'\in\mathcal{A}(s')} \widehat{Q}_k(s',a'). \tag{3}$$

In this update, no underlying model is involved, only the tuple (s,a,s',r') and the estimate $\widehat{Q}$. If $\mathcal{A}(s)$ is not known for some $s\in\mathcal{S}$, it can initially be assumed that $\mathcal{A}(s)=\mathcal{A}$. If no transition occurs when an action a is sent to the controlled system in its state s, that action is removed from $\mathcal{A}(s)$.

3.3 Q-learning for probabilistic systems

The Q-function in (1) can be interpreted as an expectation (average) of the random variable $\rho(s,a,s')+\gamma\max_{a'\in\mathcal{A}(s')} Q(s',a')$. Generally, the average m of a random variable is estimated as $\widehat{m}_k=\sum_{i=1}^{k} x_i/k$, given a set of samples x_i, $i=1\ldots,k$. To obtain a recursive formulation of this average estimate, we observe that $(k+1)\widehat{m}_{k+1}=k\widehat{m}_k+x_{k+1}+\widehat{m}_k-\widehat{m}_k$. Thus,

$$\widehat{m}_{k+1}=\widehat{m}_k+\alpha_{k+1}(x_{k+1}-\widehat{m}_k),$$

where the learning rate $\alpha_k=1/k$. Indeed, this is the stochastic approximation algorithm proposed by Robbins and Monro (1951). Replacing the average estimate $\widehat{m}_k$ with $\widehat{Q}_k(s,a)$, and the sample x_{k+1} with $r'+\gamma\max_{a'\in\mathcal{A}(s')}\widehat{Q}_k(s',a')$, gives the following update of the Q-function estimate:

$$\widehat{Q}_{k+1}(s,a)=\widehat{Q}_k(s,a)+\alpha_{k+1}\big[r'+\gamma\max_{a'\in\mathcal{A}(s')}\widehat{Q}_k(s',a')-\widehat{Q}_k(s,a)\big] \tag{4}$$

For deterministic systems, the optimal learning rate is $\alpha_{k+1}=1$, which coincides with (3).

Learning rate Generally, the learning rate α_k must satisfy the conditions $\sum_{k=0}^{\infty}\alpha_k^2<\infty$ and $\sum_{k=0}^{\infty}\alpha_k=\infty$, to be able to guarantee that the estimate $\widehat{Q}$ converges to the optimal Q-function (1). The simple learning rate $\alpha_k=1/k$, motivated by the similarity with the average estimate, fulfills the convergence conditions, but in practice it often decreases too quickly towards zero.

The following slower reductions are therefore proposed by Gosavi (2015), either $\alpha_k=\log(k)/k$, or $\alpha_k=A/(B+k)$ with for instance $A=150$ and $B=300$. Especially, the second proposal is shown to result in $\widehat{Q}$-factors close to the optimum, while the simple choice $\alpha_k=1/k$, often proposed in the literature, shows large deviations between Q and $\widehat{Q}$ (Gosavi, 2015).

Action exploration In the Q-learning procedure, the convergence of $\widehat{Q}$ towards Q is only guaranteed if all state-action pairs are updated infinitely many times. In practice, a reasonable trade-off between exploration and exploitation is obtained by the ε-greedy procedure, which is often proposed. An action is then chosen with equal but reduced probability $P_k(s)=1/k/|\mathcal{A}(s)|$ as k increases. This is done for each action $a\in\mathcal{A}(s)$ and for all states s. A greedy (asymptotically optimal) action according to (2) is then chosen with probability $1-P_k(s)$. The latter guarantees that all actions asymptotically become greedy, since $P_k(s)\to 0$ when $k\to\infty$. A common alternative to the ε-greedy procedure is to use Boltzmann exploration (Sutton and Barto, 2018).

Complex models The model-free updates of $\widehat{Q}$ for all feasible state-action pairs are assumed to be stored in a look-up table. This limits the application of this learning method to about 10^6-10^8 state-action pairs. For larger state spaces, approximate Q-functions can be used based on parametric models, for instance neural networks.

4. TEMPORAL LOGIC AND RELATED AUTOMATA

Temporal logic is often used to formulate specifications of system properties. Such formal specifications are used in model checking (Baier and Katoen, 2008), but also for synthesis of reactive systems (Pnueli and Rosner, 1989). The two most well known temporal logics are linear temporal logic (LTL) and computational tree logic (CTL).

4.1 Syntax and semantics of LTL

For a given sequence of states (a linear order), and a set AP of atomic propositions, LTL specifies when these propositions are true or false.

Definition 3. (Syntax of LTL). Given a set AP of atomic propositions, LTL formulas are defined inductively by the grammar

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1\wedge\varphi_2 \mid \bigcirc\varphi \mid \varphi_1\mathcal{U}\varphi_2,$$

where $p\in AP$. □

The additional standard connectives for propositional formulas are obtained as derived formulas. This includes $\varphi_1\vee\varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1\wedge\neg\varphi_2)$, $\varphi_1\to\varphi_2 \stackrel{\text{def}}{=} \neg\varphi_1\vee\varphi_2$, while the logical constants true and false are defined as $\top \stackrel{\text{def}}{=} p\vee\neg p$ and $\bot \stackrel{\text{def}}{=} \neg\top$. The modalities eventually and always are derived from the until operator as $\Diamond\varphi \stackrel{\text{def}}{=} \top\mathcal{U}\varphi$, $\Box\varphi \stackrel{\text{def}}{=} \neg\Diamond\neg\varphi$. These derived operators are motivated by the fact that $\top\mathcal{U}\varphi$ holds for all sequences where φ is eventually (finally) true, but before that any propositional statements are accepted. Then, φ is eventually true, and $\Diamond\varphi$ holds for all such sequences. Furthermore, $\neg\Diamond\neg\varphi$ holds for all sequences where there is no current or future time instant where $\neg\varphi$ is true. Thus, φ is true for all moments of time, and therefore $\Box\varphi$ holds.

The combined formula $\Box\Diamond\varphi$ means that $\Diamond\varphi$ will always hold. This can also be expressed as "eventually φ" is repeated forever, or in other words that φ has to be true infinitely many times. Another formulation of this fact is therefore "infinitely often φ".

The reverse combination $\Diamond\Box\varphi$ implies that eventually $\Box\varphi$ will hold, and therefore eventually "always φ" is satisfied. Alternatively, this can be expressed as eventually or finally φ will hold forever.

Definition 4. (Semantics of LTL). For a set of atomic propositions AP, consider an infinite sequence $\sigma = \sigma(0)\sigma(1)\sigma(2)\ldots$, where the value $\sigma(t)$ of the function $\sigma : \mathbb{N} \to 2^{AP}$ includes the set of all propositions $p \in AP$ that are true at time instant $t \in \mathbb{N}$. Furthermore, the infinite sequence $\sigma^k = \sigma(k)\sigma(k+1)\sigma(k+2)\ldots$ is the suffix of the sequence σ starting at time instant k. The satisfaction relation $\vDash$ between the sequence σ and an LTL formula φ is here defined inductively as (1) $\sigma \vDash p \Leftrightarrow p \in \sigma(0)$, i.e. $\sigma(0) \vDash p$, (2) $\sigma \vDash \neg\varphi \Leftrightarrow \sigma \nvDash \varphi$, (3) $\sigma \vDash \varphi_1 \wedge \varphi_2 \Leftrightarrow \sigma \vDash \varphi_1$ and $\sigma \vDash \varphi_2$, (4) $\sigma \vDash \bigcirc\varphi \Leftrightarrow \sigma^1 \vDash \varphi$, (5) $\sigma \vDash \varphi_1 \mathcal{U} \varphi_2 \Leftrightarrow (\exists k \geq 0)\, \sigma^k \vDash \varphi_2$ and $(\forall j \in [0\,..\,k-1])\, \sigma^j \vDash \varphi_1$. □

4.2 Büchi automata for LTL formulas

LTL specifies properties of infinite sequences $\sigma = \sigma(0)\sigma(1)\sigma(2)\ldots$, where the symbols $\sigma(i) \in \Sigma = 2^{AP}$ and $\sigma \in \Sigma^\omega$. Any subset $\mathcal{L} \subseteq \Sigma^\omega$, called an ω-language, can also be used to specify properties that are equivalent to LTL specifications. For instance, the formulas $\Box p$, $\Diamond q$, $p\mathcal{U}q$, and $\Box\Diamond p$ can also be expressed as the regular ω-languages p^ω, $\top^* q \top^\omega$, $p^* q \top^\omega$, and $(\top^* p)^\omega$. In the same way as a regular language for finite words can be recognized by a finite automaton, a regular ω-language can be recognized by a *Büchi automaton*.

Definition 5. (Büchi Automaton). A Büchi automaton is a tuple $B = \langle \mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_m, q_f \rangle$, where $\mathcal{Q}$ is a finite set of states, $\Sigma \subseteq 2^{AP}$ is a finite set of symbols, $\delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ is a transition function, q_0 is an initial state, $\mathcal{Q}_m$ is a set of marked (goal) states, and q_f is a forbidden state. □

The main difference between a finite automaton and a Büchi automaton is the acceptance condition, where a marked state of a finite automaton must be reached one time, while a marked state in a Büchi automaton must be reached infinitely many times. By introducing a state label, say M, for all marked states, the acceptance conditions can be expressed in LTL as $\Diamond M$ for a finite automaton and $\Box\Diamond M$ for a Büchi automaton. A marked state is indicated by a double circle. The forbidden state q_f, indicated by a cross, is included as a trap state to be reached for all non-accepted words.

An additional difference between a Büchi and a finite automaton is that a nondeterministic Büchi automaton (NBA) can not be transformed to a deterministic Büchi automaton (DBA), which is possible for a corresponding finite automaton. Since verification and synthesis of MDPs generally require DBAs, more complex but deterministic automata have been proposed. The most common example is deterministic *Rabin automata*, which can represent all LTL formulas. However, most LTL formulas of practical interest can be translated to DBAs (Alur and Torre, 2001), and in Sect. 7 another option taking care of the nondeterminism is also discussed. Hence, DBAs are assumed in this paper. For more details on the relation between LTL and Büchi automata, see Baier and Katoen (2008).

4.3 Büchi weighted product MDP

To restrict the behavior of an MDP $\mathcal{M}$ such that it satisfies an LTL formula φ, the corresponding Büchi automaton B_φ is synchronized with the MDP by a product model $\mathcal{M} \otimes B_\varphi$, sometimes called Büchi weighted product MDP or just product MDP (Baier and Katoen, 2008).

Definition 6. (MDP-Büchi product). Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, s_0, AP, \lambda, \rho \rangle$ and a Büchi automaton $B = \langle \mathcal{Q}, \Sigma, \delta, q_0, \mathcal{Q}_m, q_f \rangle$, the product MDP

$$\mathcal{M} \otimes B = \langle \mathcal{S} \times \mathcal{Q}, \mathcal{A}, P_\otimes, (s_0, \delta(q_0, \lambda(s_0))), \mathcal{S} \times \mathcal{Q}_m, \mathcal{S} \times \{q_f\}, \rho_\otimes \rangle,$$

where

(i) the transition probability

$$P_\otimes((s,q),a,(s',q')) = \begin{cases} P(s,a,s') & \text{if } q' = \delta(q,\lambda(s')), \\ 0 & \text{otherwise,} \end{cases}$$

(ii) the reward

$$\rho_\otimes(s,q) = \rho(s) + \begin{cases} \rho_M > 0 & \text{if } q \in \mathcal{Q}_m, \\ \rho_F < 0 & \text{if } q = q_f, \\ 0 & \text{otherwise.} \end{cases}$$

□

A transition $((s,q),a,(s',q'))$ in the product system $\mathcal{M} \otimes B$, determined by the transition probability $P_\otimes$, is restricted such that the state label $\lambda(s')$ of the target state s' in $\mathcal{M}$ satisfies the related transition label in B. This is achieved by the condition $q' = \delta(q, \lambda(s'))$ included in $P_\otimes$.

Forbidden states are introduced in our formulation of Büchi automata for non-accepted words, see Fig. 1. The safety specification $\Box\neg q$ in this figure specifies that no state with label q is accepted. The forbidden state 2 is therefore introduced in B_φ to model that a transition in $\mathcal{M}$ to a state with label q is not accepted.

In the reward function $\rho_\otimes$, an additional positive reward $\rho_M > 0$ is added to marked states, while a negative reward $\rho_F < 0$ is added to forbidden states. The Q-learning procedure, demonstrated in the next section, will then select actions

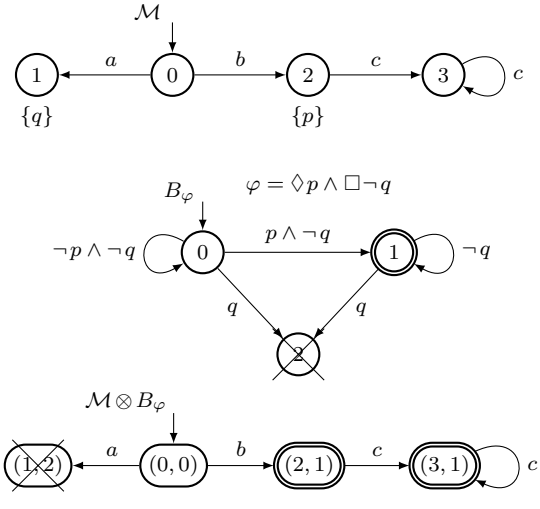

Figure 1. A (deterministic) MDP $\mathcal{M}$ has state labels p and q, and a Büchi automaton B_φ is shown for the LTL specification $\varphi = \Diamond p \wedge \Box\neg q$, including a forbidden state after transitions with the label q. The resulting product MDP $\mathcal{M} \otimes B_\varphi$ has two marked states, since the specification $\Diamond p$ is fulfilled both in state 2 and 3 in $\mathcal{M}$.

such that marked states, if possible, will be visited infinitely often. On the other hand, forbidden states will be avoided, and all together this means that a given LTL specification φ will be satisfied. When ordinary performance oriented rewards are included ($\rho \neq 0$), conflicts between the performance and temporal logic rewards may appear. This is further discussed in the next section. For the case where only temporal logic rewards are included ($\rho = 0$), it can be proven in a similar way as in (Sadigh et al., 2014; Hasanbeig et al., 2019a,b) that if there exists a policy such that φ can be satisfied with probability one, the Q-learning algorithm will also find such a policy.

5. MODEL-FREE Q-LEARNING WITH AUTOMATA SPECIFICATIONS

The model-free Q-learning procedure presented in Sect. 3, including temporal logic specifications according to Sect. 4, will now be evaluated. Some possible extensions are also discussed and illustrated.

5.1 LTL specifications

Given an LTL specification φ, the corresponding Büchi automaton B_φ is constructed. For a given state action pair (s, a), the resulting next state and reward (s', r') are determined by the controlled system. Together with an online execution of the Büchi automaton, both the next product state (s', q') and the reward can then be achieved on-the-fly. The update of the Q-function estimate according to (4) is performed when the new state and reward are available. Note that the Q-function is constructed for the product MDP. The following examples illustrate some typical specifications and results for small but nontrivial examples.

Example 1. (Liveness and safety specification). The LTL formula $\varphi = \square\lozenge p \wedge \square\lozenge q \wedge \square\neg r$ includes a liveness and fairness specification, where at least one state with label p and at least one state with label q must be visited infinitely often. At the same time a safety specification is included, specifying that no state with label r is allowed to be visited at any time. The Büchi automaton B_φ for this LTL specification is shown in Fig. 2.

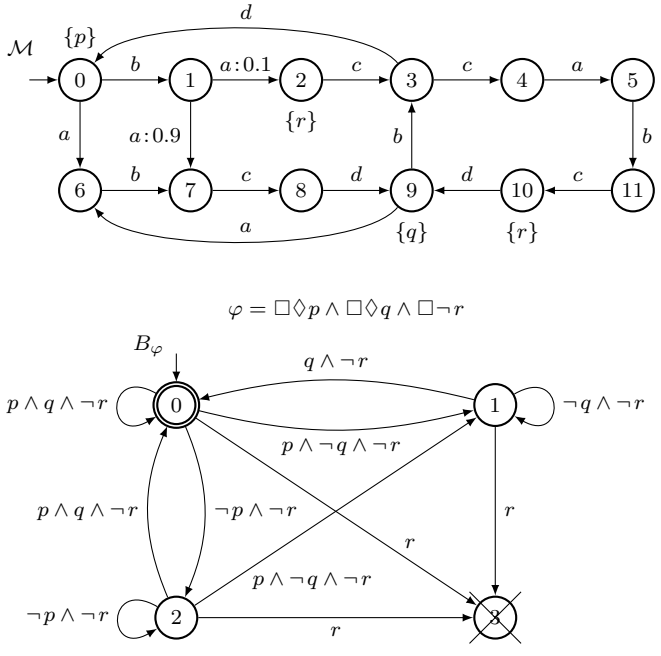

Figure 2. MPD with probabilistic uncertainty in state 1, and a Büchi automaton B_φ for the desired LTL specification $\varphi = \square\lozenge p \wedge \square\lozenge q \wedge \square\neg r$.

Applying this specification on the MDP $\mathcal{M}$ in Fig. 2 results after 1000 Q-iterations in a clear decision which action to take in the states with alternative target states. The decision is made such that the Q-function is maximized. Note the uncertainty in state 1, where there is a risk to reach the forbidden state 2. This is avoided by the resulting policy $\pi(0, 1) = a$, $\pi(9, 0) = b$, $\pi(3, 2) = d$, where the first state is the MDP state, and the second one is the B_φ state. The resulting state sequence in the MDP is the loop $0, 6, 7, 8, 9, 3, 0, \ldots$, which means that the fairness criterion is also fulfilled, where both the p and the q states are visited in every loop i.e. infinitely often. The forbidden states 2 and 10 are also always avoided. □

5.2 Alternative specifications

The basic ideas on combining Q-learning with LTL specifications, proposed by Sadigh et al. (2014); Hasanbeig et al. (2019a,b), are indeed not limited to LTL, as illustrated in the following example.

Example 2. (Regular language specification). In Fig. 3 the regular language $\mathcal{L} = (\varnothing^*\{p\}\varnothing^*\{q\})^*$ is represented by an automaton, including the forbidden behavior (repeated p or q before the alternating state label occurs). Once again, 1000 Q-iterations give a clear policy $\pi(2, 0) = b$, $\pi(2, 1) = b$, $\pi(6, 0) = c$, $\pi(6, 1) = d$, where the decision in state 2 is related to the additional rewards in the alternative transitions between state 2 and 3. In state 6 an alternation between the actions c and d occurs due to the involved automaton G. □

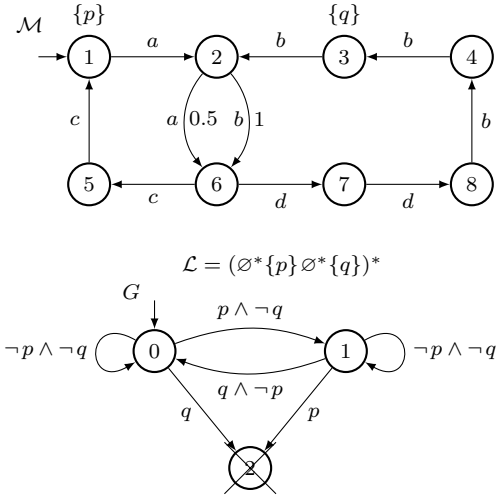

Figure 3. MPD with additional rewards in the alternative transitions between state 2 and 3. The specification is in this case a finite automaton modeling a desired alternating sequence given by the regular language $\mathcal{L} = (\varnothing^*\{p\}\varnothing^*\{q\})^*$.

Action based specifications and supervisory control Example 2 illustrates that any specification automaton with marked and/or forbidden states can be included in an MDP product with corresponding positive and negative rewards. Indeed, also specifications represented as MDPs (often without probability distributions) can be synchronized with the environment MDP, in the same way as automata specifications are synchronized with

plant models in supervisory control. In that case the transition labels in the specification are actions that restrict the behavior of the environment, instead of state labels that specify desired state label sequences, as in Figs. 2 and 3. This shows the possibility to also combine Q-learning and supervisory control, where ones again rewards are introduced on marked and forbidden states.

5.3 Safety specifications and performance rewards

Ordinary rewards are in this paper included in the term ρ in Def. 1 and from now called *performance rewards*. They are often related to time, energy and/or cost reduction. Liveness demands in temporal logic can sometimes be replaced by ordinary rewards. A desired goal state can for instance have a positive reward, and fairness between two states can be obtained by introducing rewards that are reduced if a state is visited repeated number of times. This results in more soft demands than classical liveness demands, but on the other hand it may generate much more detailed control of the fairness than just saying that every fairness state must be visited infinitely often.

Safety demands, on the other hand, are often absolute, and then logic criteria are natural. Since, there may be conflicts between liveness and performance rewards, an interesting alternative is to only have performance and safety rewards. Liveness demands are then expected to be reformulated as performance rewards. In Example 2 this situation is illustrated where a simple performance reward is included, where there is a choice between action a with reward 0.5 and action b with reward 1 in state 2 in the MDP $\mathcal{M}$ in Fig. 3. The safety condition is in this case purely determined by the automaton G in the same figure. The forbidden reward is $\rho_F = -100$, and any conflicts between the positive performance rewards in ρ and the safety demands can be avoided by selecting the negative size of ρ_F large enough. The safety demands are then considered as absolute demands, while paths among states where forbidden states can be avoided are optimized based on the performance rewards. Example 2 is a simple example of this procedure.

6. MODEL-FREE AND MODEL-BASED Q-LEARNING

So far the focus has been on model-free Q-learning. The main problem with this approach is that it generally requires a lot of data that often is not available in industrial applications. In practice, some knowledge of a system is often available. Specifically, if models are available for some parts, represented by a number of local MDPs, these models can be joined by the synchronous composition presented in Sect. 2. Assume for instance that the system is divided into two MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$, where one part say $\mathcal{M}_1$ is known at least partly. A relevant assumption is that for instance possible state transitions are known in terms of the condition $P(s, a, s') > 0$ (the next state is known but not by which probability). Then offline evaluations using the structural knowledge of $\mathcal{M}_1$ can be used to simplify the online computations, see (Baier and Katoen, 2008) Chapter 10.

Safety guarantees by local analysis As a simple illustration, consider the forbidden state 10 in the MDP $\mathcal{M}$ in Fig. 2. Going backward from this state to state 3 directly shows that only action d is possible in state 3. Evaluating the forbidden state 2 in the same model shows that there is a probability 0.1 to reach this state from state 1, and therefore state 1 must also be avoided. Therefore only decision a is possible in state 0. By this simple local evaluation, the safety demands have already been taken care of. What remains is only the fairness demand, which generally requires more global analysis.

This safety analysis can be performed in individual models, since an (extended) forbidden state can always be taken away. In the total system, the only difference can be that such states will never be reached, which means that no flexibility is removed in local handling of this safety problem.

Probabilistic transitions = uncontrollable events An interesting remark from a supervisory control point of view is that probabilistic choices alternatively can be considered as uncontrollable actions, if only safety issues are considered. In this way supervisory control theory (SCT) algorithms can be used to evaluate safety issues in a way that has not been considered in the MDP community (to the authors best knowledge).

Learn deviation between model and real system A simulation model is often available. This can be used for off-line learning, but since the model mostly includes uncertainties, an important research direction is to improve the simulation model by data from on-line measurements. The deviation between the model and the real system can then be estimated in an online learning procedure.

Some other examples of valuable knowledge that may simplify the Q-learning iteration are:

- Determine available action sets $\mathcal{A}(s)$ for all states before starting the Q-learning. Note that no alternatives $|\mathcal{A}(s)| = 1$ means that there is no decision to take, and such transitions can be abstracted either manually or by abstractions such as branching bisimulation (Lennartson and Noori-Hosseini, 2018)
- Solve the Q-learning problem offline by simulation based on available information to improve the initial value $\widehat{Q}_0$ instead of starting with no knowledge ($\widehat{Q}_0 = 0$). This is indeed a very common strategy.
- Many MDP decision problems have an optimal policy of a "threshold" type (Cassandras and Lafortune, 2008). This knowledge can reduce the number of iterations extremely much. Tests on an admission control problem shows a huge improvement in terms of number of iterations in the Q-learning procedure.

Finally, we remind again that all information that is available should be taken into account. The synchronous composition is then useful, and together with the product MDP the total system can be synchronized on-the-fly including those parts that are not known before the Q-learning procedure starts.

7. SUMMARY AND CONCLUSIONS

A brief survey of RL is given in this paper, including a short but complete derivation of the model-free Q-learning algorithm. Some recent results on how temporal logic restrictions can be included to guarantee safety and liveness properties for this model-free approach are then demonstrated. It is also shown how local analysis, based on partial information of a modular system, can be performed to improve especially the handling of safety guarantees in the optimization procedure.

An interesting question is to clarify the relation between uncontrollable events and probabilistic nondeterminism. Preliminary results show that it is possible to transform an MDP to a completely deterministic system but with an uncontrollable behavior. In that case also nondeterministic Büchi automata can be used including all LTL formulas, thus avoiding more complex automata such as Rabin and LDBA automata.

REFERENCES

Aksaray, D., Jones, A., Kong, Z., Schwager, M., and Belta, C. (2016). Q-learning for robust satisfaction of signal temporal logic specifications. In *Proc. 55th IEEE Conference on Decision and Control (CDC)*, 6565–6570. Las Vegas.

Alshiekh, M., Bloem, R., Ehlers, R., Konighofer, B., Niekum, S., and Topcu, U. (2018). Safe reinforcement learning via shielding. In *32th AAAI Conference on Artificial Intelligence (AAAI-18)*, 2669–2678.

Alur, R. and Torre, S. (2001). Deterministic generators and games for LTL fragments. In *16th IEEE Symposium on Logic in Computer Science (LICS)*, 291–300.

Astrom, K. and Wittenmark, B. (2008). *Adaptive Control*. Dover Publications, New York, second edition.

Bacci, G., Bacci, G., Larsen, K., and Mardare, R. (2013). Computing behavioral distances, compositionally. In *Mathematical Foundations of Computer Science*, volume 8087 of *Lecture Notes in Computer Science*, 74–85. Springer.

Baier, C. and Katoen, J.P. (2008). *Principles of Model Checking*. The MIT Press, Cambridge, MA.

Bertsekas, D. (2019). *Reinforcement Learning and Optimal Control*. Athena Scientific, Nashua, USA.

Busoniu, L., Babuska, R., Schutter, B.D., and Ernst, D. (2010). *Reinforcement learning and dynamic programming using function approximators*. CRC Press.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, 2nd edition.

Gosavi, A. (2015). *Simulation-Based Optimization*, volume 2nd. Springer.

Hasanbeig, M., Abate, A., and Kroening, D. (2019a). Logically-constrained reinforcement learning. Technical Report https://arxiv.org/abs/1801.08099v8, Oxford.

Hasanbeig, M., Kantaros, Y., Abate, A., Kroening, D., Pappas, G., and Lee, I. (2019b). Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. Technical Report https://arxiv.org/abs/1909.05304, Oxford.

Lennartson, B. and Noori-Hosseini, M. (2018). Visible bisimulation equivalence— a unified abstraction for temporal logic verification. In *Proc. 14th International Workshop on Discrete Event Systems, WODES'18*.

Milner, R. (1989). *Communication and Concurrency*. Series in Computer Science. Prentice-Hall.

Mohajerani, S. and Lafortune, S. (2019). Transforming opacity verification to nonblocking verification in modular systems. Technical Report https://arxiv.org/abs/1904.06242, University of Michigan.

Noori-Hosseini, M., Lennartson, B., and Hadjicostis, C.N. (2019). Incremental observer reduction applied to opacity verification and synthesis. Technical Report https://arxiv.org/abs/1812.08083, Chalmers University of Technology.

Pnueli, A. and Rosner, R. (1989). On the synthesis of an asynchronous reactive module. In *Proc. 16th International Colloq. on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, 652–671. Springer.

Poole, D.L. and Mackworth, A.K. (2017). *Artificial Intelligence — Foundations of Computational Agents*. Cambridge University Press, second edition.

Puterman, M. (2014). *Markov decision processes: Discrete stochastic dynamic programming*. John Wiley & Sons.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3), 400–407.

Sadigh, D., Kim, E., Coogan, S., Sastry, S., and Seshia, S. (2014). A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In *Proc. 53th IEEE Conference on Decision and Control (CDC)*, 1091–1096. Los Angeles.

Sutton, R. and Barto, A. (2018). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, second edition.